AD-753 403

A CLASS OF OPERATIONS SUITABLE FOR FRACTIONAL-SIZE ASSOCIATIVE MEMORIES

Donald W. Johnson

Michigan University

Prepared for:

Army Electronics Command

October 1972

AD

RESEARCH AND DEVELOPMENT TECHNICAL REPORT
ECOM-0058-61

A Class of Operations Suitable for Fractional-Size Associative Memories

Technical Report No. 61

by
D.W. Johnson

October 1972

ECOM

UNITED STATES ARMY ELECTRONICS COMMAND · FORT MONMOUTH, N.J.

CONTRACT NO. DAAB07-72-C-0058

SYSTEMS ENGINEERING LABORATORY
Department of Electrical Engineering
The University of Michigan
Ann Arbor, Michigan 48105

NOTICES

Disclaimers

Disposition

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D
(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Systems Engineering Laboratory Dept. of Electrical and Computer Engineering The Univ. of Mich., Ann Arbor, Mich. 48104	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE
A Class of Operations Suitable for Fractional-Size Associative Memories

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)
Technical Report No. 61

5 AUTHOR(S) (First name, middle initial, last name)
Donald W. Johnson

6. REPORT DATE October 1972	7a. TOTAL NO OF PAGES 53	7b. NO OF REFS 9

8a. CONTRACT OR GRANT NO DAAB07-72-C-0058	9a. ORIGINATOR'S REPORT NUMBER(S) 010749-5-T
b. PROJECT NO. C8-2-08501-01-C8-CA	
c.	9b OTHER REPORT NO(S) (Any other numbers that may be assigned this report) ✓ ECOM-0058-61
d.	

10 DISTRIBUTION STATEMENT
Approved for public release; distribution unlimited

11 SUPPLEMENTARY NOTES	12 SPONSORING MILITARY ACTIVITY U.S. Army Electronics Command Fort Monmouth, New Jersey 07703 Attn: AMSEL-WL-S

13 ABSTRACT
Associative memories have extremely useful capabilities, but the memories are extremely expensive. One way of circumventing the high hardware cost is to use an associative memory which is smaller than the data base, and process the data by pages. By using a smaller memory the hardware costs are thus reduced. Some operations can be performed quite efficiently on an associative memory smaller than the data base, (Fractional-Size Associative Memory) while others cannot. In this report a class of operations which are performed efficiently on a Fractional-Size Associative Memory is defined.

I

DD FORM 1 NOV 65 1473 UNCLASSIFIED

Reports Control Symbol
OSD-1366

THE UNIVERSITY OF MICHIGAN

SYSTEMS ENGINEERING LABORATORY

Department of Electrical and Computer Engineering
College of Engineering

SEL Technical Report No. 61

A CLASS OF OPERATIONS SUITABLE FOR FRACTIONAL-SIZE ASSOCIATIVE MEMORIES

by

D. W. Johnson

October 1972

This research was supported by U.S. Army
Electronics Command Contract DAAB07-72-C-0058

ABSTRACT

Associative memories have extremely useful capabilities [illegible] the memories are extremely expensive. One way of circumventing the high hardware cost is to use an associative memory which is smaller than the data base, and process the data by pages. By using a smaller memory the hardware costs are thus reduced. Some operations can be performed quite efficiently on an associative memory smaller than the data base, (Fractional-Size Associative Memory) while others cannot. In this report a class of operations which are performed efficiently on a Fractional-Size Associative Memory is defined.

1.0 Introduction

In the study and design of large scale systems, problems of data management and data processing are often encountered. One area of such problems concerns the selection of efficient hardware for a system, given that a known set of operations are to be performed on the selected hardware. This is, of course, one of the primary goals of a systems designer. In order to select the proper hardware for a system, however, a designer must know for which type of operations each hardware component is best suited. In this report the problem of determining the suitability of operations is considered for the particular case of associative memories.

In the period of time since associative memories were first designed and built, numerous studies [e.g., 1-7] have been made to determine which operations are well suited to their use. Although these studies have shown associative memories to be powerful and versatile, the application of associative memories has been quite limited, due to their high cost. Since the cost of these memories is proportional to the number of bits, it is logical to use as small a memory as will provide adequate performance. The issue is, therefore, to determine how small a memory can be used and still obtain satisfactory performance. Intuitively, it seems clear that the size memory required would depend on the operation to be performed and the size of the data base. In particular, one operation might perform well on a memory which is smaller than the data base, while a second operation might perform very poorly under these conditions. It would therefore be advantageous to know what are the characteristics of the first operation which allow it to perform well in a situation which is totally unsuited for the second operation. This is the issue which is discussed in the remainder of this report.

The goal here will be to determine which operations are well suited to an associative memory which can hold only some fraction of the data base required for those operations. The reason for this is quite simple. In many instances the cost of an associative memory which is large enough to contain the entire data base is prohibitively high. If, however, an associative memory, which can hold some fraction, α, of this data base, can be employed without paying too high a cost in increased processing time, then the capabilities of the associative memory may be available within the price range of the user. (In the next section an example will further clarify this idea.) As will be shown later in this report, certain operations are suitable to such a reduced associative memory size while others are not. To determine which operations are suitable the following steps are taken in this report.

First, the concept of associative memory is briefly reviewed and the assumptions concerning the data processing environment, which are the basis of the analysis in this report, are presented. Second, a precise meaning is given to the term "operation", as used in this report. Next, a definition of what is meant by an operation being suitable for a full-size associative memory is presented. With this definition as a basis, the characterization of operations suitable for associative memories which must process data bases larger than themselves (Fractional-Size Associative Memory) is begun. One conclusion reached is that the basic associative search capabilities (instructions) of an associative memory must fit the characterization chosen. The final characterization is formalized as the definition of a class of operations which are suitable for Fractional-Size Associative Memories. Finally, examples of members of this class are discussed.

1.1 Motivational Example

In this section a situation is examined in which a Fractional-Size Associative Memory can be used efficiently. For the sake of brevity and clarity the example will be kept as simple as possible.

The situation is the following. A large data base is frequently queried as to the contents of that data base. To answer these queries a computer is used. However, the data base is so large that a random access memory large enough to store the entire data base is too expensive. Also, other demands made upon the computing system would force the data base to be removed from core periodically. It has been determined that the computer system, with only the random access capability of a conventional memory, could not keep up with the demands on the system. The queries to the data base are of the type that an associative memory resolves quite efficiently. For instance, a typical query would be to find all words in the data base whose values lie within a given range of values. A full size associative memory can simultaneously, for all data in the memory, ascertain which data meet this condition[1]. With a random access memory only one word may be checked at a time. Since the queries are suitable for the associative memory the queries could easily be processed if the memory were large enough to hold the entire data. This cannot be done since the cost would be far too great, much greater than for the same size conventional memory. To summarize, the situation is as depicted in Figure 1. Time and cost[2] constraints are present as indicated by the horizontal and vertical dotted lines, respectively. (The size of the data base is assumed to be fixed.) The crosshatched area indicates the region in which the system must operate in order that its performance be acceptable. The circled dots indicate the time-cost coordinates of the associative and conventional memories which are large enough to hold the entire data

[1]See Appendix A for an example of how this query could be resolved by an associative memory.

[2]Although cost is a criterion in this example it is not used in the more general problem of interest in the remainder of this report. Reasons for this are discussed in Section 4.

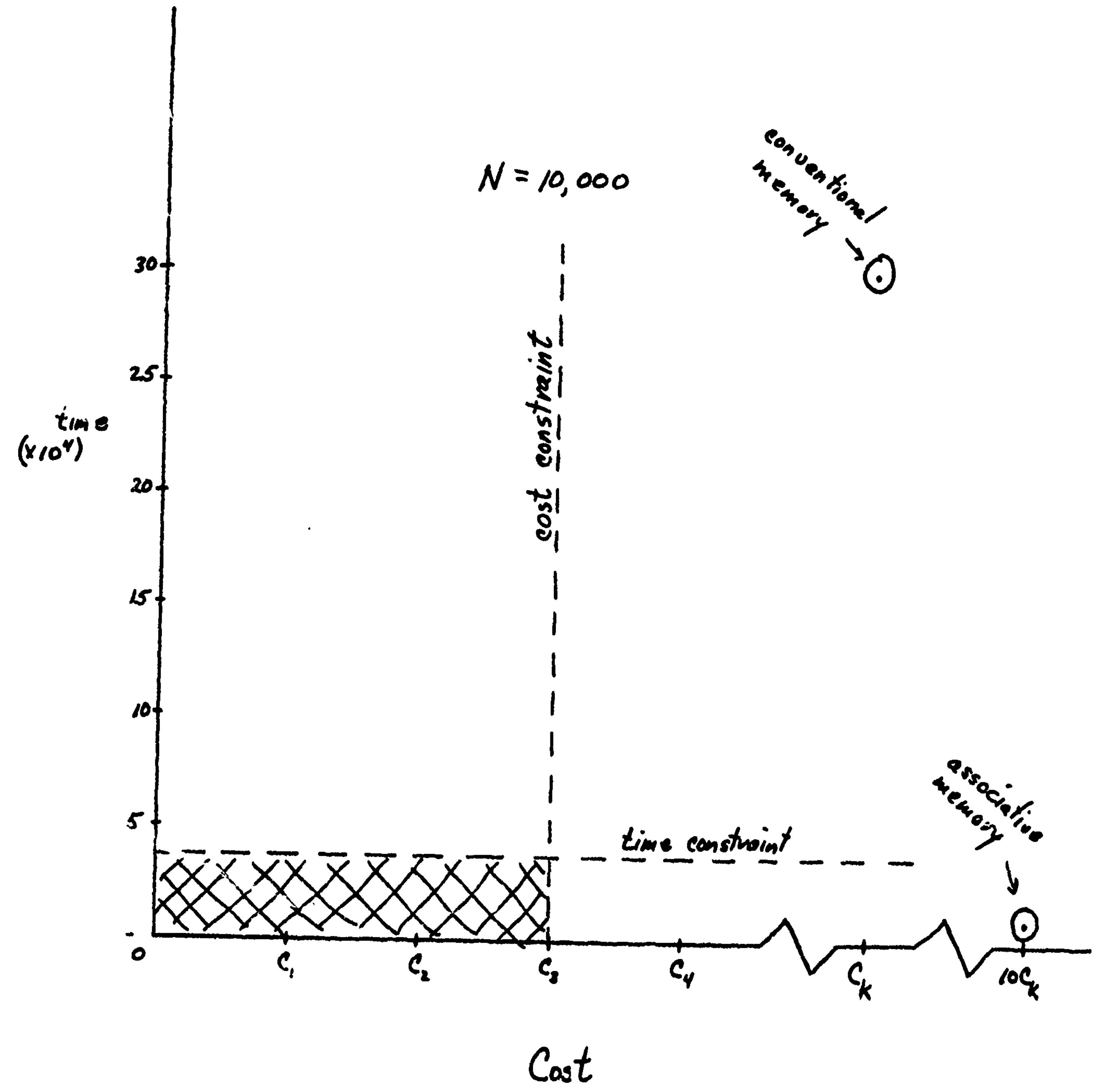


Figure 1 Time and Cost Constraints For Example System

base. The conventional memory meets neither of the two constraints, while the associative memory meets only the time constraint.

Since the associative memory does meet the time constraint, one way of solving this problem is to reduce the cost of the associative memory to an acceptable level by reducing its size. If this smaller associative memory, which meets the cost constraint, can still meet the time constraint, then a solution has been found. To show that this can be done in some instances, Appendix A includes programs which implement the example operation described above, using a conventional memory and an associative memory. These are analyzed to determine how much time the associative memory will require to perform the operation, as a function of the memory size. The results are presented in Figure 2. As can be seen there is, indeed, a range of sizes for the associative memory such that both constraints are met, as evidenced by the part of the curve which passes through the cross-hatched area. Thus, for the operation described above, it is clear that a Fractional-Size Associative Memory can be used efficiently. Indeed, it constitutes the only solution to the problem, as presented.

This example leads into the question of how to determine when Fractional-Size Associative Memories can be used efficiently. For some operations such a memory can be used efficiently while for others it cannot. For example, ordering a list on some parameter is somewhat awkward on a smaller memory, i.e., one which cannot hold all the data. For this reason it would be desirable to know what characteristics can guarantee that a given operation is suitable for a Fractional-Size Associative Memory. To attack the problem of finding these characteristics some groundwork must be prepared. In particular, precise notions of suitability and operation must be defined, and the problem to be discussed must be posed in much more specific terms. In the next sections this groundwork is completed. This includes a review of the concept of an associative memory so as to avoid possible confusion on the part of the reader when specific terms are used.

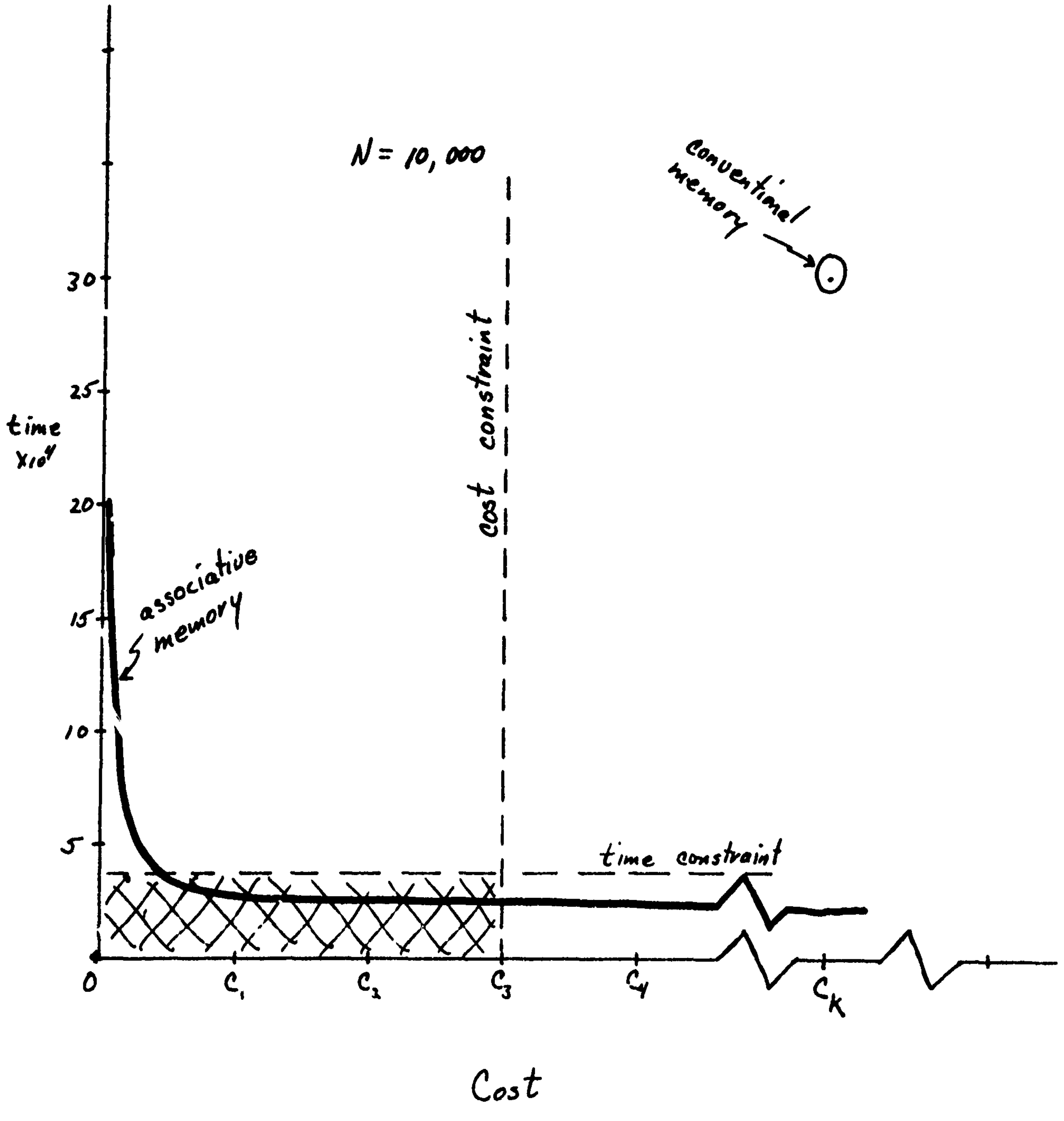


Figure 2 Example of cost-time Trade-off for Small Associative Memory

2.0 The Associative Memory System and Environment

2.1 The Basic Associative Memory

Since the rest of this report will presume a certain familiarity with associative memories a brief review of the basic structure of an associative memory will now be presented.

In brief, any memory which has the capability to retrieve data by contents may be termed as associative memory. Technically, the term content-addressable is a more apt description of this type of memory, but the term associative memory seems to be commonly accepted and will be used throughout this report.

The configuration which is usually considered minimal, in terms of hardware, for an associative memory is shown in Figure 3. It consists of a collection of storage elements which are grouped into logical units known as memory registers. These registers may be subdivided into fields. Also required are three types of registers known as comparand register(s), mask register(s), and response register(s). These four component types, along with control logic, allow the basic function of content addressability to be performed. As an example of how these components are used, suppose it is desired to determine which words[1] have Field A equal to 100. The procedure would be first, set Field A of the mask register to its "ON" condition (all other Fields are "OFF")

[1]The unit of information held by one memory register is known as a word.

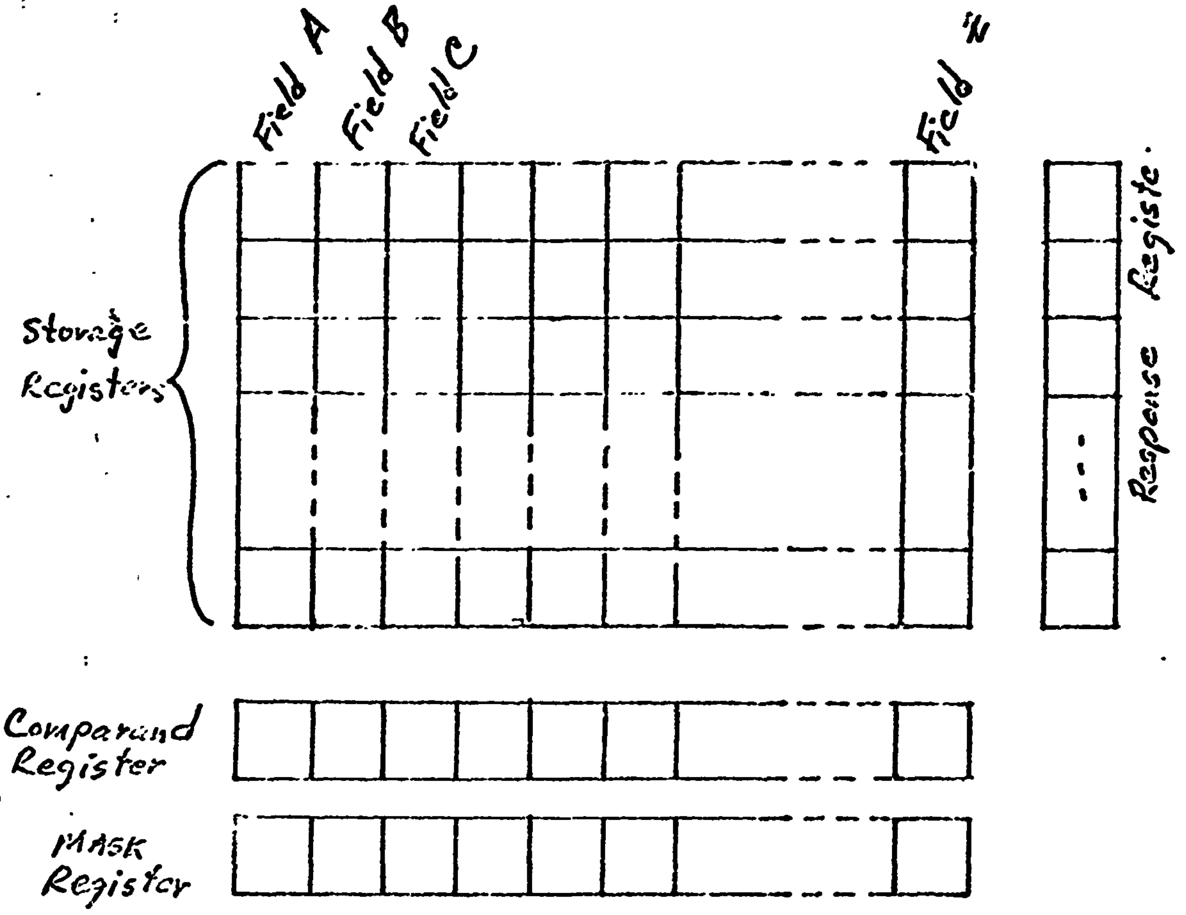


Figure 3 Simple Associative Memory

to indicate a search is to be performed on Field A, only. Next, the value 100 is placed in Field A of the comparand register to signify the value to be searched for. Then, the "EXACT MATCH" search instruction is executed and, afterwards, the response register indicates those words which meet the search criterion.

2.2 Capabilities of Associative Memories

Even though a memory system with the capability described above is a very powerful device, other capabilities are generally added as well. Typical capabilities are the following [1,5].

1. EXACT MATCH - finds all words for which the bits indicated by the mask match the corresponding bits in the comparand
2. MAX - finds the location of the maximum value in memory for those bits indicated by mask register
3. MIN - analogous to MAX
4. LESS THAN - finds all words for which the bits indicated by the mask have value less than the value of the corresponding bits in the comparand
5. GREATER THAN - analogous to LESS THAN
6. LESS THAN OR EQUAL - same as LESS THAN but indicates words with equal value as well
7. GREATER-THAN-OR-EQUAL - analogous to LESS THAN OR EQUAL

8. **NEXT-HIGHER-THAN - finds word(s) having the smallest value of indicated bits which is greater than the value of those bits in the comparand**

9. **NEXT-LOWER-THAN - analogous to NEXT HIGHER THAN**

10. **BETWEEN COMPARANDS - finds those words for which the value of the indicated bits is between the values of the corresponding bits of two comparands (This operation requires two comparand registers)**

2.3 The Associative Memory System Model

A great deal of variation exists among present and proposed associative memory systems in terms of hardware available, instruction set, and the configuration of the memory system. Therefore, it is appropriate at this point to present the memory configuration and the instruction set upon which the remainder of this report will be based.

The memory configuration is depicted in Figure 4. The arrows between the various components indicate paths of data flow. The data is passed among the memory levels by high speed, dedicated channels. All data transfers, including those to and from the associative registers, are parallel-by-bit, serial-by-word transfers. The buffer memory is present to mask latency delays characteristic of disk, drum, and other bulk storage. The goal is to anticipate requests for data by the associative memory and have that data waiting in the buffer when it is needed. The size of the buffer will not be discussed, but it will be assumed that

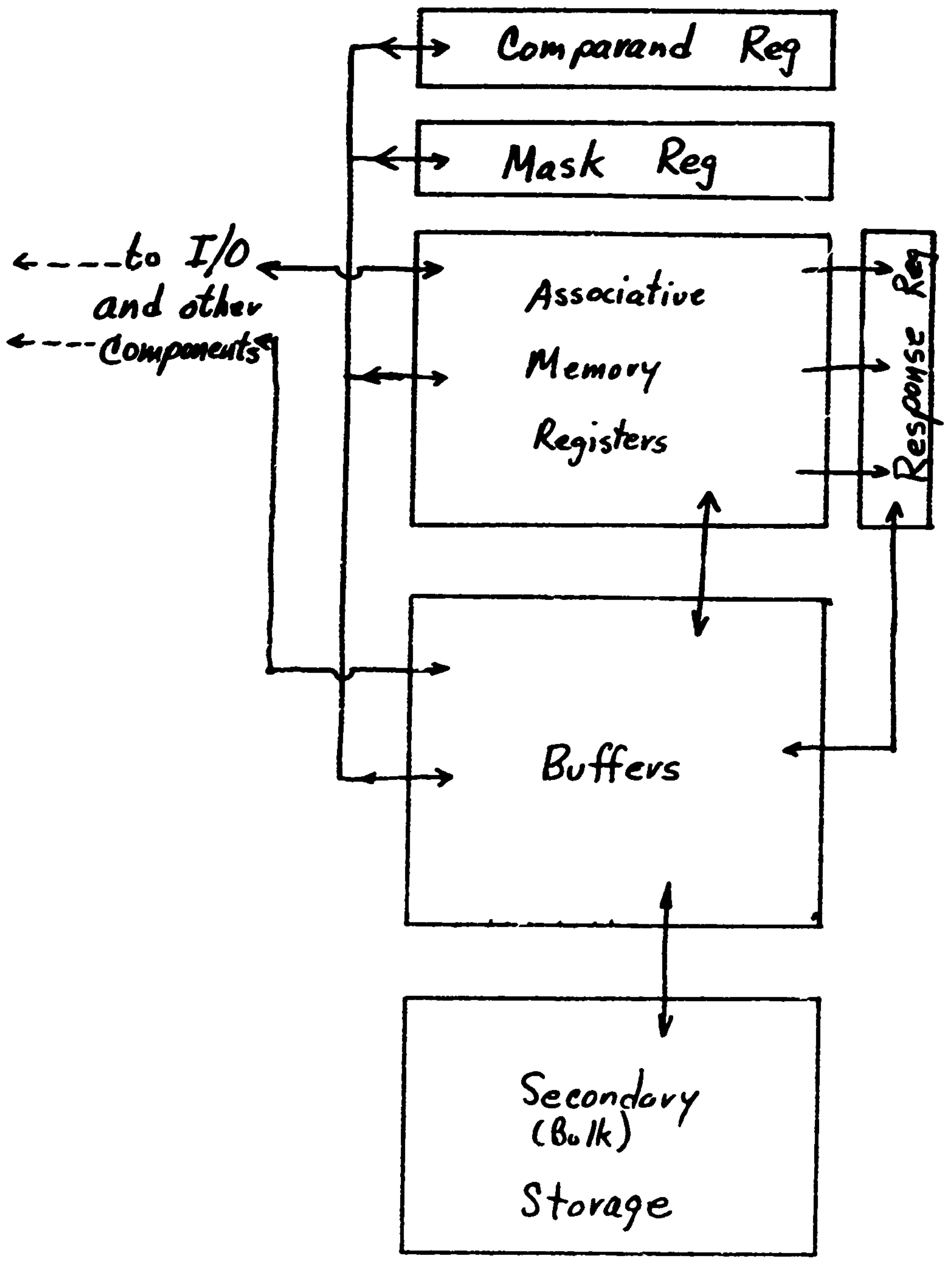


Figure 4 Memory Model for Report

the buffer is larger than the associative memory. The dotted channel arrows going to the left indicate a data path to other components of the computer system. These may include conventional memory, I/O units, or other registers. These will be of no concern, except where specifically mentioned later on. No particular length will be assumed for the associative registers or the associative memory registers, except in specific examples. It will be assumed, unless otherwise noted, that the response register contains one bit for each word in the associative memory.

The instruction set of the associative memory will include five basic search commands, plus commands to move data to and from the associative memory, plus other "housekeeping" instructions.

The search instructions are the following

1. EXACT MATCH
2. MAX
3. MIN
4. LESS THAN
5. GREATER THAN

For each of these five instructions the words which do not satisfy the search criteria are indicated at the completion of the instruction by the correspondence bit of the response register being set to zero. The bits corresponding to words which meet the search criterion are left unchanged. Other instructions which are assumed to be part of the instruction set are the following.

1. LOAD - causes the specified register to be loaded from the specified location (the register and the location are parameters of the instruction).

2. SET RESPONSE - causes response register to be set to all ones or all zeros, whichever is specified.

3. COMPLEMENT RESPONSE - causes each bit of response register to be complemented.

4. COUNT RESPONDERS - causes a count of the number of bits set to zero in the response register to be taken.

5. MOVE RESPONSE - causes those words for which the corresponding bits in the response register are zero to be written into the buffer memory or into some other specified place.

6. WRITE RESPONSE - causes a specified symbol to be written into the set of bits designated by the mark register, for all locations indicated by the response register.

7. LOAD RESPONSE - causes those locations indicated by response register to be loaded from specified region.

For an operative system other instructions would be required, in addition to the seven listed above. These instructions are presented only to give some insight to the range of capabilities which is assumed for the associative memory system in the remainder of this report. They are not meant to be an exhaustive list.

One bit of notation which will be used throughout the remainder of this report is the symbol "α". This symbol will designate the size of the associative memory relative to the size of the data base to be processed (i.e., $0 < \alpha \leq 1$). For example, if the data base has 10,000 words and the associative memory can hold 100 words, then $\alpha = \frac{100}{10,000} = .01$. This is the reason for the term Fractional-Size Associative Memory.

3.0 Operations

Up to this point the term "operation" has been used on an intuitive basis. The discussion in this section will provide a more precise definition of the term in the context of this report.

An operation is an abstract concept which is similar to the concept of "goal". That is, an operation is a specification of a desired end result with no specification of the manner in which the result is to be achieved. For example, "Find the maximum value of field A", is an operation. The actual achievement of the goal, i.e. finding the maximum value of field A, can be accomplished in various manners. More formally, an operation may be thought of as a mapping which maps data into output.

Structurally these operations may be viewed as having been generated by a set of primitive, or typical, operations. For any given system this set of typical operations corresponds to the instruction set of that system. Operations not included in these typical operations may be constructed by building an appropriate sequence of the typical operations. For example, suppose the instruction set for some system consisted of the two groups of instructions discussed in Section 2.3, and that it is desired to perform the following operation. "Of those words which have (Field A) $<$ 100 and (Field B) $>$ 10, find those words for which (Field C) = 1." This operation is not a member of the typical set of operations but it could be constructed from the following sequence. (It is assumed, for simplicity, that the data has already been loaded.)

1. LOAD Comparand[1] - loads test values into Fields A, B, and C.
2. LOAD Mask - sets Field A to "on" state, other fields to "off".
3. Set Response - puts all 1's in response sequence.
4. LESS THAN - finds words with (Field A) $<$ 100 RESPONSE bits for all others are set to zero.
5. LOAD Mask - sets Field B to "on" state, others to "off".
6. GREATER THAN - response bits for all words having (Field B) $>$ 10 are set to zero.
7. LOAD Mask - set Field C to "on" state, others to "off".
8. EQUAL TO - after this operation response bits still set to one indicate those words which meet all three criteria.

[1]In practice, all LOAD commands would require a specification of where to load from, but this is a detail which is unimportant to the discussion above.

4.0 Suitability of Operations for Fractional-Size Associative Memories

The goal of this section is the development of the definition for the Minimum Suitable Class of Operations. This definition will specify sufficient conditions for an operation to be suitable for a Fractional-Size Associative Memory. In other words, the Minimum Suitable Class will be a subclass of those operations which are suitable for the Fractional-Size Associative Memory. The first step toward this goal will be to define suitability.

4.1 Suitability

The word "suitable" has, unfortunately, different intuitive meanings to different people. To one person an operation which is suitable for an associative memory is one which can be performed by that type of memory. To someone else it may mean that the operation can be performed within a specified time using the associative memory. To another person it may mean that the operation can be performed more rapidly, or more cheaply, or both in an associative memory as compared to a conventional memory. Here suitability of an operation for an associative memory will be defined relative to time to perform that operation using a conventional memory. The definition will say nothing about the cost of the associative memory, for two reasons. First, the money available to spend on an associative memory is normally a constraint peculiar to each data processing system. Thus, an absolute bound on the cost would be arbitrary and probably useless. Second, the ratio of the cost of an associative memory to the cost of a conventional memory depends on the technologies used to build each of them. Therefore, constraining the cost of an associative memory relative to the cost of a conventional memory wou.d likely prove to be of little value either.

The definition will characterize suitability in terms of full size associative memories and full size conventional memories, i.e., memories which are large enough to hold the entire data base.

> <u>Definition</u> An operation, f, is suitable for a full size <u>associative</u> memory A, if, for any implementation of f on a conventional memory, C, an implementation of f on A can be found which is faster.

In this definition an implementation is a program. The definition says that f can be performed faster using A than is possible using C.

Note that this definition is for full size associative memories. It should be clear to the reader that operations which are suitable under this definition may, if implemented in a Fractional-Size Associative Memory, require more processing time than if implemented in a conventional memory. In general, this will depend upon the size of the Fractional-Size Associative Memory and the nature of the operation. Since the size the Fractional-Size Associative Memory is normally limited by economic considerations only the nature of operations will be considered here.

The goal, then, is to characterize the Minimum Suitable Class of Operations as described above. The only question is where to start.

One starting point is to consider what are the most essential, or basic, capabilities of associative memories. Clearly, it is the associative search capabilities, such as MAX, EXACT MATCH, etc. These are the capabilities which distinguish associative from conventional memories. Any class of operations which are to be suitable to a Fractional-Size Associative Memory should contain these capabilities. For this reason, the nature of these operations will be studied as the basis of the characterization of the Minimum Suitable Class of Operations. Recall that in Section 2.3 five associative operations were listed, and it was stated that these five could be combined to implement the capabilities of almost any other set of associative search instructions.

4.2 Characterization of the Basic Associative Operations

The five associative search operations listed in Section 2.3 are repeated below for convenience.

1. EXACT MATCH
2. GREATER THAN
3. LESS THAN
4. MAX
5. MIN

The five operations corresponding to these instructions will hereafter be referred to as the "basic associative operations". For example, the operation corresponding to EXACT MATCH is that mapping which maps the input data set to those members of the input data which match some predetermined constant.

The first step in characterizing these operations will be to define the range and domain of the mapoings. To do this some notation must first be defined.

Notation

The basic object of interest is the single word, or datum. Therefore, let

$$\mathcal{W}_n = \left\{ \text{n-bit words} \right\}$$

where $n = 1, 2, \ldots,$.

Now the largest unit of information which will be of interest is the data base. The set, $\mathcal{D}$, of all data bases is defined as

$$\mathcal{D} = \left\{ d \mid d = (w_1, \ldots, w_k), \quad k = 1, 2, \ldots, \text{ and } w_i \in W_n \right\}$$

where the word length n is assumed fixed.

Then the general form of the mapping of interest is

$$f : \mathcal{D} \longrightarrow \mathcal{D}$$

i.e., f maps k-tuples (data bases) to j-tuples.

Thus, we now have the set

$$S_f = \left\{ f \mid f : \mathcal{D} \longrightarrow \mathcal{D} \right\}$$

of all mappings of $\mathcal{D}$ into itself. S_f contains the Minimum Suitable Class of Operations. To determine the desired subset of S_f further restrictions must be developed.

Since suitability is concerned with the processing time of operations, it seems logical that restrictions to S_f should reflect time in some sense. To obtain the desired restrictions, therefore, it is necessary to determine which characteristics of the five basic associative instructions make them "more suitable", with respect to processing time, for a Fractional-Size Associative Memory.

Processing time is composed of two parts, namely, data movement time (e.g., loading and unloading of data) and execution time (i.e., the time required to execute instructions). In an associative memory the time required to load or unload one datum is generally equivalent to the time required to execute an associative search instruction, such as MAX. Therefore, it is desirable to load and unload the data as few times as is possible. In Appendix B it is shown that the basic associative operations can be implemented on a Fractional-Size Associative Memory without loading any datum more than once, i.e., no reloading of data is required. Thus, one restriction on the members of the Minimum Suitable Class should be that they each have an implementation such that only a single pass through the data is required to compute the output. In the next section such a restriction is discussed in terms of the familiar concept of causality.

The execution time required for some implementation, $\mathcal{P}_f$, of an operation f is governed by two factors. First is the time required to execute the implementation (or program) on the data in a given associative memory. Second is the numbers of times the $\mathcal{P}_f$ must be repeated to process the data base of interest. Thus, both of these quantities must be bounded in some manner. In the preceeding paragraph it was pointed out that $\mathcal{P}_f$ should not require any data to be reloaded. However, no constraints were placed on how many data could be retained in the memory between applications of $\mathcal{P}_f$. For instance, if $\mathcal{P}_f$ allows only one new datum to be brought in between each application of $\mathcal{P}_f$, then it is, in some sense[1], "less suitable" than if it brought all new data in after each application. In Appendix B it is demonstrated that all of the five basic associative operations have implementations such that no more than one datum is retained between each application of the respective implementations. This characteristic of the five basic associative operations should also be reflected in a restriction on the members of the Minimum Suitable Class. This is also discussed in the next section.

In terms of execution time the important fact to be noticed about the five basic associative operations is the following. Each of these five operations, f, has an implementation $\mathcal{P}_f$ such that the execution time[2] of a single application of $\mathcal{P}_f$ is proportional to $\frac{1}{M-1}\ t(\mathcal{P}_{f_{cm}})$ where $t(\mathcal{P}_{f_{cm}})$ is the time to execute $\mathcal{P}_{f_{cm}}$, which is the fastest implementation of f using a conventional memory and M is the number of

[1]In the sense that $\mathcal{P}_f$ would be applied more times in the first instances than in the second instance.

[2]See Appendix B

registers in the associative memory. Again, this type of processing time relationship is reflected in a constraint on the members of the Minimum Suitable Class, which will be discussed below.

4.3 Formalization of Physical Constraints

In the previous section the characteristics of the basic associative operations were discussed. In this section these characteristics will be used to develop the definition of the Minimum Suitable Class of Operations. The discussion will start at the hardware level and progress to the level of mappings.

In Figure 5 the hardware constraints discussed previously are represented. The associative memory consists of M words. M-1 words of data are read into the memory, processed, and some results may be printed out as M words or fewer. The input and output processes are represented as one-way tapes to denote the no-reloading-of-data constraint.

The single word shown separated from the rest of the words represents the capability of retaining one word of information to characterize data previously processed. The one word of information may also be referred to as the "state" of the computation. Since the word has n bits the computation may be in any of 2^n states at any given instance. Figure 5 illustrates a physical realization of the first two constraints discussed earlier. Now it remains to characterize these constraints in terms of the mapping $f : \mathscr{D} \longrightarrow \mathscr{D}$.

To begin the characterization of these constraints it is first noted that the use of a Fractional-Size Associative Memory requires that the

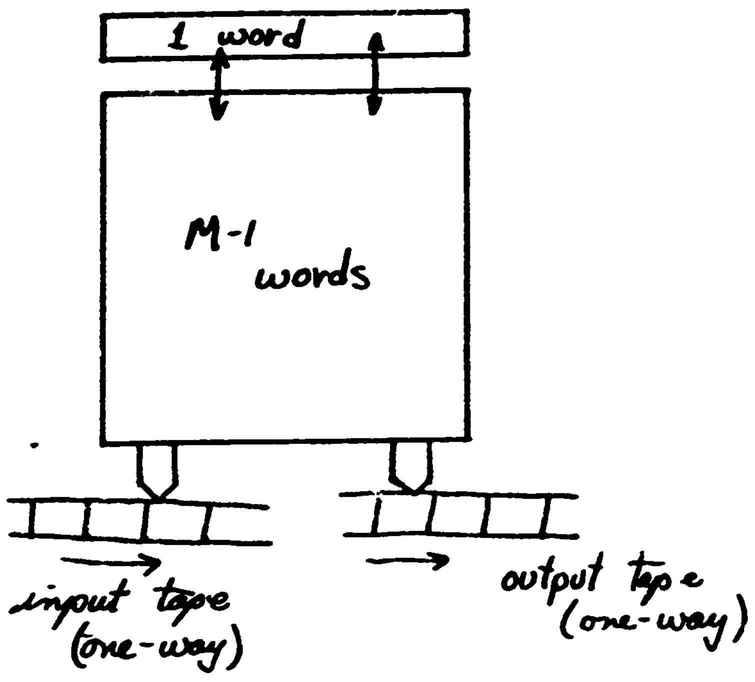


Figure 5 The Machine To Be Characterized

data base, $d \in \mathcal{D}$, must be processed in pages, $p_1, p_2, \ldots, p_k$, of M-1 words. This fact will be acknowledged by writing $d = p_1 p_2 \ldots p_k$, where $p_1 p_2 \ldots p_k$ is the concatenation of the individual pages. Note that by representing d in this fashion a time structure has been placed on the pages by the corresponding indices. Page p_1 is processed first, page p_2 is processed second, and so forth. This is the type of time structure and page-wise orientation of data that is present in the real situations which are of interest. In the sequel these facts will be kept in mind by writing, with some slight abuse of notation, $f(d) = f(p_1 p_2 \ldots p_k)$, where the p_i's all have the same number of words.

Just as a time structure exists on the input so does one exist on the output. In terms of Figure 5, a page, p_1, of input is read, and an output y_1 is produced; a page p_2 is read and y_2 is produced, etc. When all of the input, $p_1 \ldots p_k$, has been produced, then the output tape will contain $y_1 \ldots y_k$. The time structure present here must also be reflected in the mapping f. This will be accomplished by constraining f to have the following characteristics.

$$f(p_1) = y_1 \qquad \ell g(y_1) \leq M$$

$$f(p_1 p_2) = y_1 y_2 = f(p_1) y_2 \qquad \ell g(y_2) \leq M$$

$$\vdots$$

$$f(p_1 \ldots p_k) = y_1 \ldots y_k = f(p_1 \ldots p_{k-1}) y_k \qquad \ell g(y_k) \leq M$$

where $\ell g(y_i)$ is the length of y_i, i.e., the number of words of output. The length constraint reflects the fact that the memory has M words, and the output is contained in these words when the computations are finished.

By introducing this time structure on d and on f(d) as shown above, f has been constrained to be a causal mapping, i.e., the output $y_1 y_2 \ldots y_i$ is dependent only on $p_1, p_2, \ldots, p_i$. In addition, the sequential output behavior expected of the model in Figure 5 has been captured. This guarantees that an implementation for f exists which does not require any reloading of data.

The above restriction guarantees that f is a causal mapping and does not require more than M words out at a time. This means that a state-oriented procedure for realizing f can be derived[1], although an infinite number of states may be required. To illustrate this, first define the mapping r'_k ("right end" mapping), $r'_k : \mathcal{D} \longrightarrow \mathcal{D}$, such that

$$r'_k (f(p_1 \ldots p_k B)) = r'_k (f(p_1 \ldots p_k)\, U) = U \tag{2}$$

where $U, B \epsilon \mathcal{D}$ and $k = 0, 1, 2, 3, \ldots$

With the aid of the mapping r'_k, the partition π_f of   is defined as follows

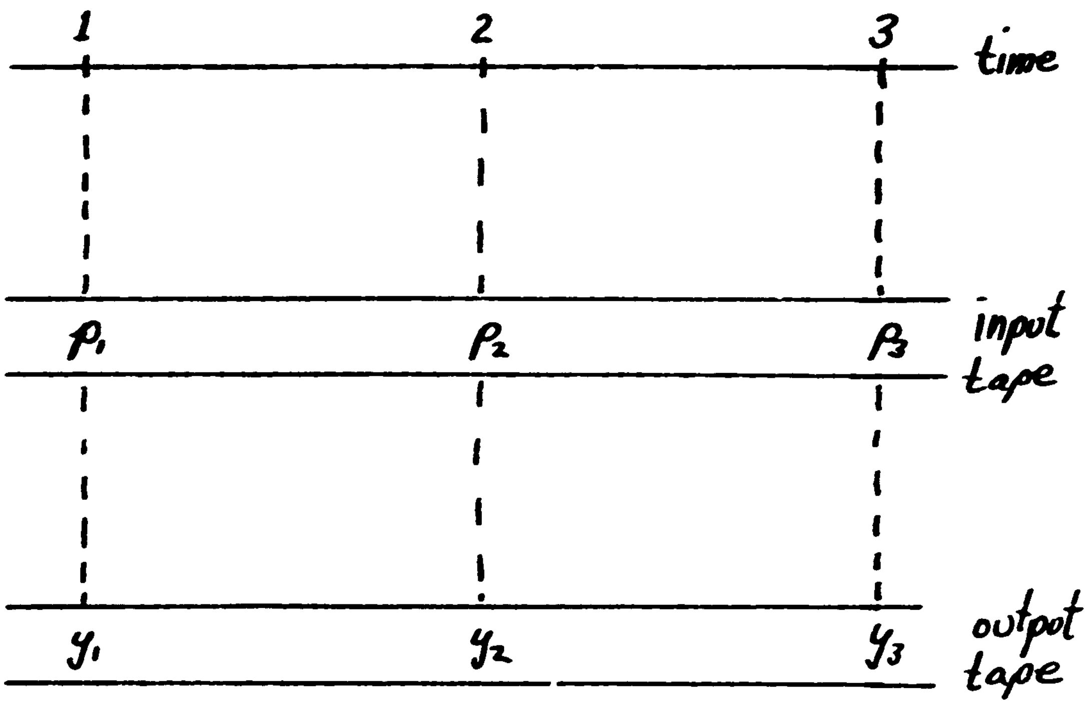
$$(p_1 \ldots p_i) \overset{\pi_f}{\sim} (p'_1 \ldots p'_s) \Longleftrightarrow r'_i f\,(p_1 \ldots p_i B) = r'_s f\,(p'_1 \ldots p'_s B)$$

for all B in $\mathcal{D}$

[1]This is a classic problem in automata theory. See, for example, [8], page 344.

This partition is the basis of forming the Nerode machine [8] in classical sequential machine theory, and it is well known that the number of states required by the Nerode machine which realizes f is equal to the number of equivalence classes (blocks) in π_f. In other words, if the partition π_f has R blocks then the Nerode machine which realizes f has R states.

Now recall that the machine in Figure 6 was constrained to hold the state of the computation in a single n-bit word, i.e., only 2^n states are allowed. It is clear, therefore, that the type of functions, f, that are of interest are those which can be realized by a 2^n state Nerode machine. Alternatively, the partition π_f of $\mathcal{P}$ must have no more than 2^n equivalence classes, or blocks. This constraint along with the causality and length constraints of equations (1) are sufficient to allow the machine of Figure 5 to compute f. Figure 6 gives some insight to the behavior of this machine as a function of time. At time 1 the page p_1 is read into the machine and the output is $f(p_1) = y_1$. At time 2 the input is p_2 and the output is y_2. On the output tape is the sequence $y_1y_2 = f(p_1p_2)$ as it should be. Similarly, at time 3, p_3 is the input, and y_3 is the output. The output tape then contains $y_1y_2y_3 = f(p_1p_2p_3)$. This example illustrates only the output behavior of the machine and not the state behavior. However, it is sufficient to allow consideration of the following situation. Suppose at time 1 the machine reads p_1 from the input tape and writes y_1 on the output tape. At time 2 the input is



$$f(\rho_1) = y_1$$

$$f(\rho_1 \rho_2) = y_1 y_2$$

$$f(\rho_1 \rho_2 \rho_3) = y_1 y_2 y_3$$

Figure 6 Output Behavior of Machine

p_2 and, on the basis of this new input, the machine decides that the output tape should read y_2, not y_1y_2. The machine has "changed its mind", so to speak, and decided that y_1 is not to be part of the output. The output tape moves in one direction only, so the previous output, y_1, cannot be erased. The only solution is to put out some special symbol, e, which means "disregard previous output", followed by y_2. This situation is depicted in Figure 7. This type of behavior by the machine will be shown to be very useful. Unfortunately, the constraints placed on f, above, are too restrictive to allow such behavior since, in Figure 7, $f(p_1p_2) = y_2 \neq f(p_1)y_2$. Therefore, the constraints will be altered slightly as follows. The function to be constrained will be denoted as g ($g: \mathcal{P} \rightarrow \mathcal{Y}$) to avoid confusion with f, above. Since the constraint on the behavior of g is important below, it will be formalized in a definition.

<u>Definition</u> A function g is behaviorally suitable or <u>b-suitable</u> if

$$\begin{array}{lll}
g(p_1) = y_1 & & \ell g(y_1) \leq M \\
g(p_1p_2) = \begin{cases} g(p_1)y_2 \\ \text{otherwise} \\ y_2' \end{cases} & & \begin{array}{l} \ell g(y_2) \leq M \\ \ell g(y_2') \leq M \end{array} \\
\vdots & & \\
g(p_1 \ldots p_k) = \begin{cases} g(p_1 \ldots p_{k-1})\, y_k \\ \text{otherwise} \\ y_k' \end{cases} & & \begin{array}{l} \ell g(y_k) \leq M \\ \ell g(y_k') \leq M \end{array}
\end{array} \qquad (3)$$

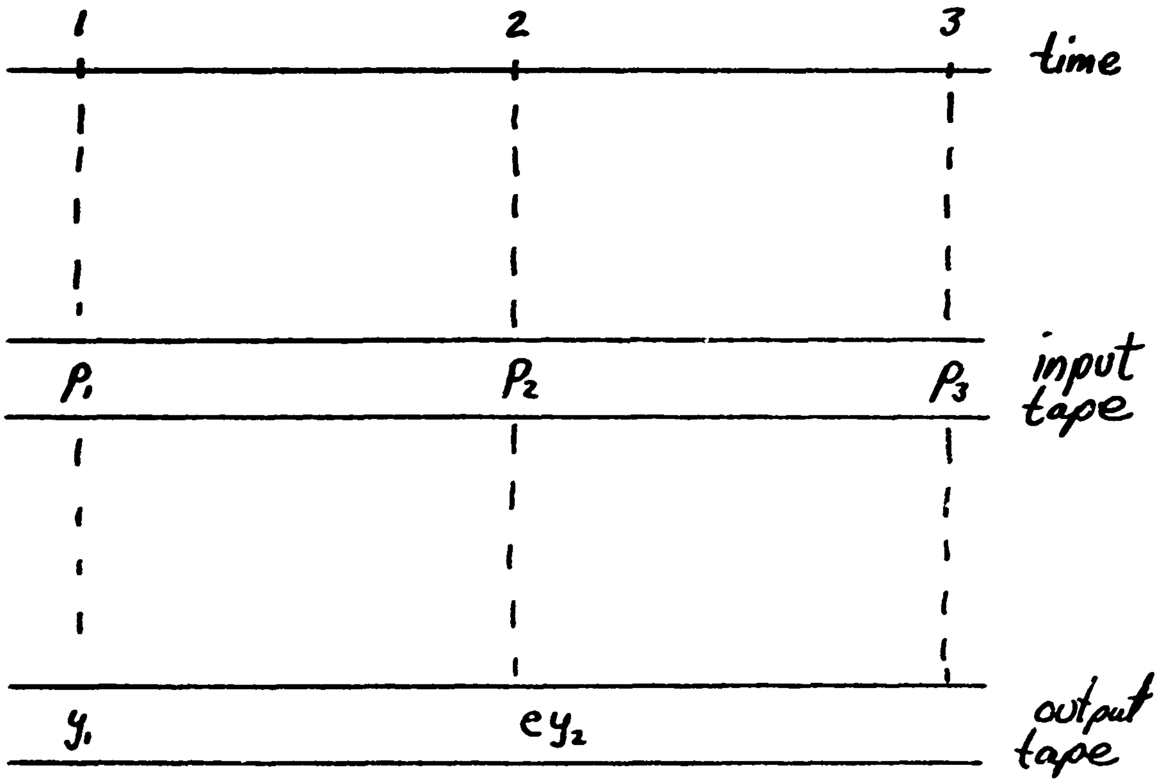


Figure 7 Situation Where Machine "Changed Its Mind"

This constraint is satisfied for any function, g, for which

$$g(p_1 \ldots p_k) = \begin{cases} y_1 \ldots y_k \\ \text{or} \\ y_2 \ldots y_k \\ \text{or} \\ \vdots \\ y_k \end{cases}$$

Clearly, f, from above, satisfies this constraint. However, the behavior illustrated in Figure 8 is also acceptable now. For instance, if $g(p_1 \ldots p_k) = y_k$, then at time k the machine merely puts ey_k on the output tape. This recall, means to disregard all output in the past, i.e., the desired result is the output generated after the symbol e, namely y_k.

Given this new, weaker constraint it is still necessary to impose the cardinality constraint on the partition of g. A new "right end" operator r_k must be defined, however, because of new constraint above. Therefore, define $r_k: \mathcal{P} \longrightarrow \mathcal{P}$ as follows

$$r_k(g(p_1 \ldots p_k B)) = \begin{array}{l} u \ \text{ if } g\,(p_1 \ldots p_k B) = g(p_1 \ldots p_k)\,u \\ \text{otherwise} \\ u' \ \text{ if } g(p_1 \ldots p_k B) = u' \end{array} \qquad (4)$$

Now the partition π_g can be defined similarly to π_f, using r_k rather than r'_k.

$$p_1 \ldots p_k \overset{\pi_g}{\approx} p'_1 \ldots p'_s \iff r_k g(p_1 \ldots p_k B) = r_s g(p'_1 \ldots p'_s B)$$
$$\forall B \in \mathcal{P}$$

as with π_f, the number of blocks in π_g must be no greater than 2^n. This guarantees that a realization of g exists which has a suitable number of states. This is formalized as follows.

<u>Definition</u> A function, g, is state suitable or <u>s-suitable</u> if the cardinality of π_g is no greater than 2^n.

It is worthwhile here to mention the dependence of the constraints of g on M, the associative memory size, and on n, the word length. It is clear that functions exist which will satisfy the constraints for some particular values of M, n but not for others. For instance, the partition of D depends on M but not on n. However, the cardinality constraint on the partition is a function of n. Therefore, a great enough reduction in n can make almost any function unsatisfactory.

Two of the three physical constraints discussed in the previous section have now been formalized as constraints on mappings. The third constraint, which directly involves the execution time of an implementation of some operation, will now be treated as follows. It was stated earlier that each of the five basic associative operations had an implementation such that $t(\mathcal{P}_{f_{am}}) \sim \frac{1}{M-1} t(\mathcal{P}_{f_{cm}})$. Therefore, this fact will be used as an additional constraint on g. This is formalized by the following definition.

<u>Definition</u> An implementation of g, $\mathcal{P}_{g_{am}}$, is t-suitable if

$$t(\mathcal{P}_{g_{am}}) \leq \frac{\beta}{M-1} t(\mathcal{P}_{g_{cm}})$$

where β is some finite constant, and M is the size of the associative memory in words. This merely says that, as the page size increases, the execution time for the implementation $\mathcal{P}_{g_{am}}$ grows at least a power of M more slowly than does the conventional memory implementation.

4.4 Definition of the Minimum Suitable Class

From the results in the previous section the Minimum Suitable Class can now be defined. Recall that this class is to contain those operations which are best suited to Fractional-Size Associative Memories.

Definition A function, g, is a member of the Minimum Suitable Class if

1. g is b-suitable
2. g is s-suitable
3. There exists an implementation of g, $\mathcal{P}_g$, such that $\mathcal{P}_g$ is t-suitable.

As an exercise it will now be demonstrated that the operations for MAX, and LESS THAN meet these conditions. First, consider the operation for LESS THAN, denoted as "ℓt". The state at any given time is arbitrary, since no information about past data is required to process future data. Therefore, ℓt is trivially s-suitable since the state need never be changed. In terms of behavior ℓt can be represented as

$$\ell t(p_1 \ldots p_k) = \ell t(p_1)\, \ell t(p_2) \ldots \ell t(p_k)$$

$$= \ell t(p_1 \ldots p_{k-1})\, y_k$$

Thus, ℓt is b-suitable, and in Appendix B an implementation $\mathcal{P}_{\ell t}$ is presented which demonstrates ℓt to be t-suitable. Therefore, as was already known, ℓt is a member of the Minimum Suitable Class.

Now consider the operation for MAX, denoted "max". For this operation a state value is required. In particular, to process future data correctly it is necessary to know the maximum value encountered in data already processed. This value may be called the state of the computation. Since this value is obtained as data it is clear that it will always fit into one word of memory. Thus, no more than 2^n possible states exist. Thus, max is s-suitable. To show that max is b-suitable consider the following.

$$\max(p_1 \ldots p_k) = \max(\max(p_1 \ldots p_{k-1}) p_k)$$

Here the need for the special symbol "e" for the machine can be seen and, consequently, the need for the mapping g rather than f. The past output, $\max(p_1 \ldots p_{k-1})$ (which is also the state of the computation), becomes incorrect whenever a larger value is found on p_k and, even if no larger value is found, a redundant value would be placed on the output tape. Therefore, when a page is read in, say p_k, the output is $e(\max(\max(p_1 \ldots p_{k-1}) p_k))$. Since this value is always a single word the length constraint is satisfied and, thus, max is b-suitable. Again, Appendix B contains an implementation $\mathcal{P}_{\max}$ which shows that max is t-suitable. Therefore, all three conditions of membership for the Minimum Suitable Class of Operations have been met.

5.0 Example Members of the Minimum Suitable Class

In this section two commonly used operations are considered. It is shown that these operations are members of the Minimum Suitable Class of Operations.

5.1 Example One (Identification)

The first operation which will be considered is known by various names, such as identification, classification, grouping, etc. The specific operation of concern for this example is the following. There are g group specifications (Figure 8). Each specification consists of a range of values for each of n parameters. Each datum in the data base has n + 1 fields. The first n fields contain parameter values, and the n + 1st field is used to hold a code word which designates to which group the datum belongs. A datum is said to belong to a particular group of the value of each of its n parameters falls within the range of values allowed for that parameter by that group. For example, the datum shown in Figure 9 can be put into group 2 since each parameter value falls within the range specified for that parameter for group 2.

It will now be demonstrated that identification is a member of the Minimum Suitable Class of Operations. The operation will be devoted by "ident.". Note that the only result of applying ident is to place the proper code word in the n + 1st field of each data word. This is done on the basis of the values of the fields of that datum, only.

group	Parameter Ranges P_1	P_2	P_3	$\cdots$	P_n
1	$a_{11}-b_{11}$	$a_{12}-b_{12}$	$a_{13}-b_{13}$	$\cdots$	$a_{1n}-b_{1n}$
2	$a_{21}-b_{21}$	$a_{22}-b_{22}$	$a_{23}-b_{23}$		
$\vdots$	$\vdots$				
g	$a_{g1}-b_{g1}$	$a_{g2}-b_{g2}$	$a_{g3}-b_{g3}$		$a_{gn}-b_{gn}$

data format

φ_1	φ_2	$\cdots$	φ_n	code

Figure 8 Formats for Data and Group Specification for Example One

	Parameter Values				
Group	P_1	P_2	P_3	P_4	P_5
1	1-4	1-6	7-8	9	2-9
2	1-6	2-3	1-10	5-7	9-11
3	7-12	1-2	10	6	5-10
4	1	5-6	4-5	1-5	3-7

example datum

5	2	9	6	9	

Figure 9 Example of Group Specification

Thus, the operation may be written as

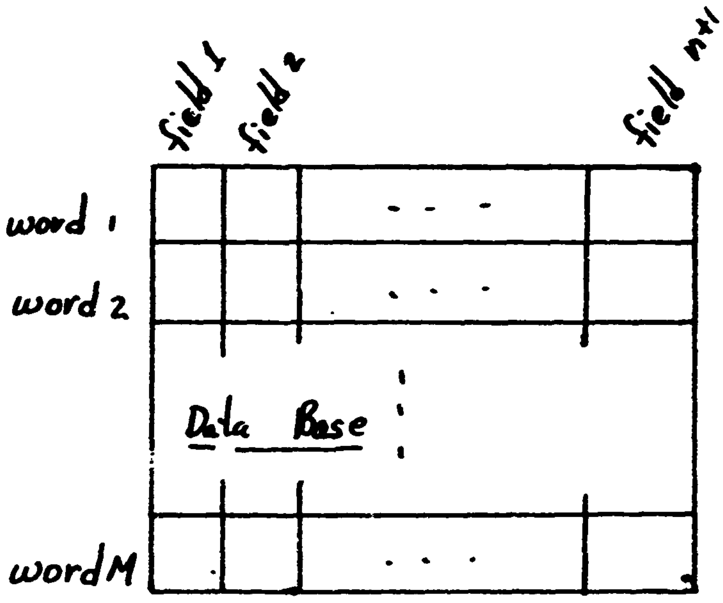
$$\text{ident}\,(p_1 \ldots p_k) = \text{ident}\,(p_1 \ldots p_{k-1})\ \text{ident}\,(p_k)$$

The length constraint for behavioral suitability is satisfied since the same data goes out as was read in, except for the code word. Thus, ident is b-suitable. Also, as stated above the code word for each datum depends only on the value of the fields of that datum, so no information about data previously processed is required. Therefore, no state information is kept and the operation is s-suitable.

To show that ident is t-suitable requires deeper consideration of the operation. This involves some estimate of the execution time required by ident on both associative and conventional memories, as a function of M. Several assumptions will be made in the process of obtaining this estimate. The general attitude taken is to make assumptions which favor the conventional memory over the associative memory. The estimate for the conventional memory will be obtained first.

5.1.1 Conventional Memory Execution Time Estimate

For the conventional memory time estimate it will be assumed that the data structures of Figure 10 are being used. The data to be identified are stored in a simple linear sequence of words, as are the group definitions. The group definitions, however, are assumed to be ordered on the value of a_{i1}. This will allow some modified form of a binary



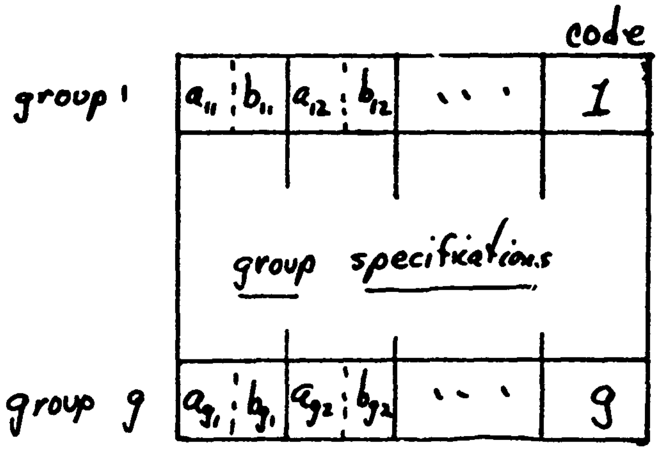


Figure 10 Data Structures Used For Time Estimate With Conventional Memory

search to be performed in order to find those group definitions which include the field 1 value of a given datum.

The procedure will be to first find those definitions whose range for field 1 values (a_{i1}, b_{i1}) includes the value of field 1 for that datum, ℓ, under consideration. The next n-1 field values of ℓ are then compared to the appropriate ranges of these selected definitions. That group specification for which all ranges include the corresponding field values of ℓ is defined to be the group to which ℓ belongs. The code for this group is placed in field $n + 1$ of ℓ. The process is then repeated for each of the other data in the data base.

Since the time to execute an algorithm such as that just discussed is dependent upon the speed of the particular machine used, an indicator of this time will be used, instead. The indicator is the number of compare instructions executed. Two compare instructions are needed for each step in the binary search and two compares are needed to check each of the n-1 fields remaining. It is well known that the binary search will require $\ell og_2 g$ probes to search an ordered list of g elements. Thus, $2\, \ell og_2 g$ compares are required for the first parameter.

Since it is not known, in general, how many groups may be selected by this binary search, it is conservatively assumed that only one group is selected. Thus, only $2(n-1)$ additional compares will be required (to check the remaining n-1 fields). The total number of comparisons is, therefore, $[\, 2\ell og_2 g + 2(n-1)] \cdot M$. It is clear that, for M large with respect to n and g, the number of comparisons is proportional to M.

5.1.2 Time Estimate For Processing One Page of Data in Fractional-Size Associative Memory

An associative memory such as that described in Section 2.3 can identify each datum in a page (m-1 words) of memory as follows.[1] First, perform a GREATER THAN and a LESS THAN operation using as comparands the a_{11}, and b_{11} values for group 1. This will find all those data whose first parameter value lies within the range of values acceptable by group 1. These data will be indicated by the corresponding bits remaining "1"s in the response register (See Section 2.3) while other data have their associated bits set to a zero. Now these same two instructions are repeated, but values a_{12}, b_{12} are used as comparands this time. This step eliminates those data whose first parameter values are acceptable but whose second parameter values are not. The associated response register bits for these data are set to zero.

This prccess is repeated for the remaining parameter value ranges for group 1. When this process has been completed, those data for which the associated response registers bit is a "1" are marked as members of group 1. The entire process is then repeated for group 2, group 3, ..., group g.

This process is clearly independent of M. The execution time is proportional to the number of GREATER THAN and LESS THAN instructions which are executed. In this example 2ng such instructions are required.

---

[1] Assume the response register has been set to all "1's" before processing begins.

Therefore, the execution time is proportional to ng, which is a fixed constant for a given problem.

5.1.3 Comparison of Time Estimates

The time estimates above indicate that the execution time, Te, for identification is of the form $Te_1 = K_1 M$ for conventional memory and of the form $Te_2 = K_2$ for associative memroy, where K_1, K_2 are constants with respect to M. Thus, it can now be stated that identification is t-suitable, since

$$Te_2 = \frac{K_2}{K_1} \frac{Te_1}{M} = \frac{\beta}{M} Te_1$$

as required by the definition of time suitability.[1] It has now been shown that the identification operation meets the three requirements for membership in the Minimum Suitable Class of Operations.

5.2 Example Two - (Index Table Look-up)

The situation of interest for this example is depicted in Figure 11. There is a large data base which contains information to be retrieved upon request. This retrieval is accomplished with the aid of the index table and the associative memory. The index table is a directory, each entry of which is a sequence of descriptors the last one of which is an address. The address is the location in the data base of the information described by the descriptors. A retrieval request is

[1]Clearly, this satisfies the definition of t-suitability since $\frac{\beta}{M} < \frac{\beta}{M-1}$.

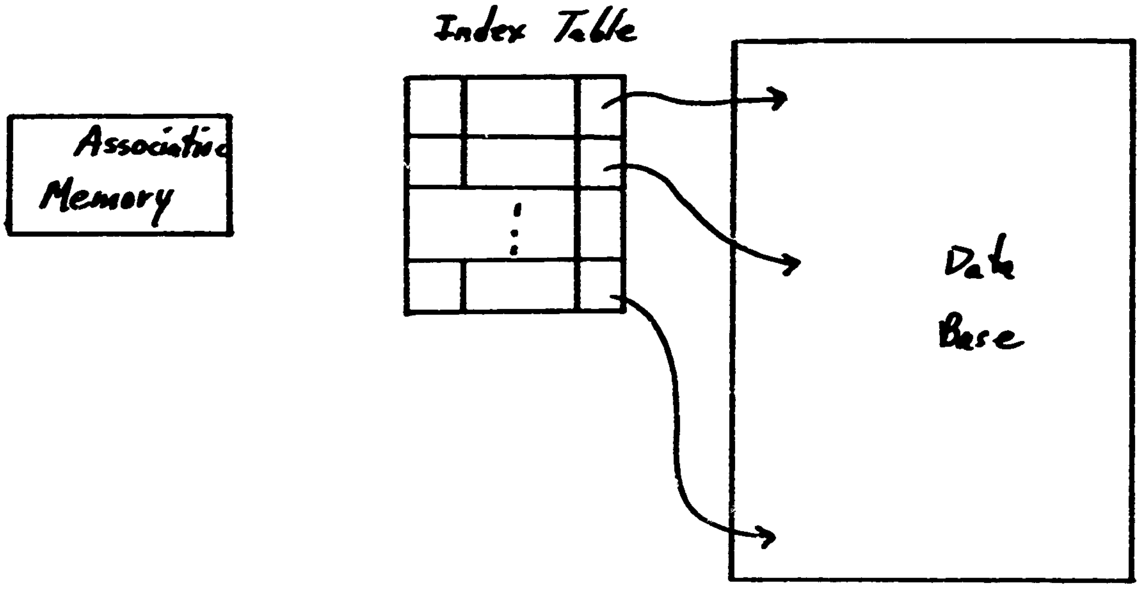


Figure 11 Simple Index Table for Information Retrieval

presented as a list of descriptors. Using the associative memory this list of descriptors is compared against those in the directory. When a match is found, the address is used to retrieve the desired information. This operation of mapping the index table and list of descriptors to an address will be denoted by "I_x" in the sequel. This operation is extremely important because of the almost universal need for data retrieval of the general type described above. It will now be shown that I_x is a member of the Minimum Suitable Class and, therefore, is suitable to a Fractional-Size Associative Memory.

Clearly, the index table, together with the input descriptors, is an element of the set θ described earlier. Also, the output descriptor is an element of θ. Therefore, I_x is an operator on θ, I_x: $\theta \longrightarrow \theta$.

In the discussion above it was stated that the process of indexing is basically a comparison between the list of input descriptors and the individual entries in the index table. This comparison between the input list and any given entry is independent of any other entries. From this, it should be clear that any collection of the entries in the index table may be processed in an associative memory independently of the remainder of the table. This is signified as follows.

$$I_x(p_1,\ldots,p_k) = I_x(p_1)\, I_x(p_2)\ldots I_x(p_k)$$

The operation I_x is, therefore, b-suitable. I_x is also s-suitable since, as in the previous example, no information need be kept about data previously processed.

It can also be shown in a manner similar to that of the previous example that I_x is t-suitable. The details will not be presented here since they would add little information and are lengthy. The operation I_x has now been shown to be a member of the Minimum Suitable Class.

6.0 Summary

In this report the problem of determining which data processing operations can be efficiently performed on Fractional-Size Associative Memories has been studied. The study proceeded in the following manner.

It was argued that certain operations are better suited to associative memories than are other operations. Therefore, a decision was made to determine which operations were best suited and, then, study the common characteristics of those operations. It was then argued that the associative search operations, i.e., those operations implemented by the associative search instructions, were the most suitable operations for Fractional-Size Associative Memories. Five operations which encompass the associative search capabilities of almost all associative memories were then chosen for detailed study. The results of this study was a characterization of these operations in functional terms. This characterization was used to define the Minimum Suitable Class of Operations. This class of operations is claimed to contain those operations which are most suitable for implementation on a Fractional-Size Associative Memory.

REFERENCES

1. Fulmer, L.C. and W. C. Meilander, "A Modular Plated-Wire Associative Processor," Goodyear Aerospace Corp., GER-14727, March 1970.

2. Fuller, R. H., "Content-Addressable Memory Systems," University of California, Los Angeles, Report No. 63-25, June 1963.

3. Green, R. S., J. Minker, and W. E. Shindle, "Analysis of Small Associative Memories for Data Storage and Retrieval Systems," Technical Report RADC-TR-65-397 Vol. 2, 1966, AD 489 661.

4. Rogers, J. L., and A. Wollinsky, "Associative Memory Algorithms and Their Cryogenic Implementation," AD 429 521.

5. Baker, F. T., and W. E. Triest, "Advanced Computer Organization," IBM, Technical Report No. RADC-TR-66-148, 1966, AD 484 444.

6. Bird, R. M., J. L. Cass, and R. H. Fuller, "Study of Associative Processing Techniques," AD 800 387.

7. Lee, F. F., "Study of 'Look-aside' Memory," IEEE Trans. on EC, Vol. C-18, Nov. 1969.

8. "IBM System/360 Model 65 Functional Characteristics," IBM Systems Reference Library, File No. 5360-01, Form A 22-6884-0.

9. Peskin, A. M., "Associative Capabilities for Mass Storage Through Array Organization," AFIPS Conference Proceedings, Vol. 37, 1970 FJCC.

APPENDIX A

Details For Motivational Example

In this appendix programs are exhibited for an associative memory and a conventional memory which implement the operation discussed in the example of Section 1.1. Various assumptions have been made to simplify each of these programs. These will be introduced as they are needed. Recall, the goal sought in the example is to determine all those numbers in some data base which lie within a given range of values, i.e., those numbers which are greater than some number, A_1, and less than some number, A_2. First the program for the conventional memory will be presented.

A.1 Conventional Memcry Program

For the conventional memory the following type of algorithm will be represented by the program below. The values A_1 and A_2 will be held in two registers. The programs starts by testing the first data word to determine if it is greater than A_1. Then, it is tested to determine if it is less than A_2. If both conditions are satisfied this datum is marked or moved to a special place. Then this procedure is repeated on the remaining data words.

After each instruction in the program a number is given to indicate the estimated number of machine cycles which would be needed to implement that instruction. These estimates are derived from characteristics of the IBM 360/65 [8].

Program

Compare A_1, d	7	
branch decision	5	
Compare A_2, d	7	
branch decision	5	
move or mark data	17	(only if data meets both tests)
branch to start over	6	

The number of cycles required to process N data words is, therefore,

$$T_{N_1} = 30N + 17C$$

where C is the number of words marked. Now, to obtain the results of the example of Section 1.1 let N = 10,000, and C = .1N. Thus,

$$T_N \cong 32 \times 10^4 \text{ cycles.}$$

A.2 Associative Memory Program

The program for the associative memory is almost identical to the program for the conventional memory except that the associative memory can perform many comparisons simultaneously. Therefore, this program is somewhat less complex than the one for the conventional memory.

Program[1]

Set Response Register	2	set to all 1's
load comparand	6	load A_1
GREATER THAN	42	comparison
Load comparand	6	load A_2
LESS THAN	42	comparison
move accepted words	17/word	

[1]The cycle estimates for the associative search operations were taken from [1]. The estimates are based on a 32 bit word. Other cycle times were estimated to be comparable with those for the conventional memory.

The number of cycles required to process one memory load (a page) of data is 98 + 17/correct word. For the data base this is

$$T_{N_2} = 98K + 17C$$

where K is the number of pages of data in the data base, and C is the number of words accepted by the program. Now, let C = .1N and N = 10,000, as for the conventional memory, and obtain

$$T_{N_2} = 98K + 1.7 \times 10^4 \text{ cycles.}$$

This is the equation plotted in Figure 2 of Section 1.1.

A.3 Memory Cost Estimates

Individual cost estimates were not derived for each type of memory. Instead, an estimate was made for the relative costs of the two types of memories. This, of course, is heavily influenced by the type of technology employed. For the example of Section 1.1 the a cost ratio of 10:1 (associative: conventional) [9] was assumed.

APPENDIX B

Implementations for the Basic Associative Operations

In this appendix implementations for the basic associative search operations of Section 2.3 are exhibited. These implementations have the characteristics discussed in Section 4 in that they satisfy the following two conditions (hereafter referred to as conditions 1 and 2).

1. When the data base is processed a part (page) at a time, then no more than one word of data may be retained from previous processing when a new page of data is transferred into the memory. This is the one word "state" discussed in Section 4.
2. The execution time required by these implementations, when applied to one page of data, is proportional to $\frac{1}{M}$ $(t(\mathcal{P}_{f_{cm}}))$, for large M, when $t(\mathcal{P}_{f_{cm}})$ is the execution time required for the fastest implementation of that operation on a conventional memory.

The implementations for an associative memory are presented in the form of algorithms. These are analyzed to obtain some insight concerning the execution time of these algorithms. Then estimates are obtained for executing the operation using a conventional memory, and the two estimates are compared.

B.1 Algorithms and Time Estimates

First the algorithm for the "max" operation will be presented.

0. Preset the "state" of the computation.

1. Load a new page of data into the associative memory.
2. Set response register to all 1's.
3. Execute the MAX instruction.
4. Read new page into memory, saving only the state. In this case the state is that word containing the maximum value from the previous page.
5. Repeat steps 2-5 until the data base has been completely processed.

This algorithm demonstrates that a one word "state" description is sufficient to contain the necessary information from one page of data to the next. Thus, condition 1 is satisfied. It also demonstrates that the execution time[1], for one page of data, is a constant with respect to the size of the memory.

Now consider how this operation would be implemented in a conventional memory. The important aspect to be considered is that no more than two words may take part in a COMPARE operation in the typical conventional memory type of computer. Thus, the usual method employed to find the maximum value of a list of words is to compare two words at a time and keep the one larger in value to compare against the third word, and so forth. This requires M-1 comparisons to be performed. Thus, the execution time is proportional to M-1.

---

[1]Recall that the execution time does not include the time required to load the memory.

From the discussion above it follows that the ratio of the execution times for implementations of the associative memories and the conventional memories is

$$\frac{t(\mathcal{P}_{max'am})}{t(\mathcal{P}'_{max'cm})} = \frac{K_1}{M-1} \quad \text{where } K_1 \text{ is a constant}$$

or

$$t(\mathcal{P}'_{max'am}) = \frac{K_1}{M-1} \cdot t(\mathcal{P}'_{max'cm})$$

Condition 2 is, therefore, satisfied.

The discussion for the "min" operation follows the same path as given above for max. For this reason it will not be presented here. The "greater than", or "gt" operation will be discussed next.

GREATER THAN, LESS THAN, EXACT MATCH

The algorithm for implementing gt on a Fractional-Size Associative Memory is the following.

1. Load a new page of data into the memory.
2. Set Response register to all 1's.
3. Execute the GREATER THAN instruction.
4. Move all data for which the corresponding bit in the response register is a 1 to some special area.
5. Repeat steps 1-4 until data base has been processed.

Two points are immediately evident for this algorithm. First, no information is retained from one page of data to the next, i.e., no state information is required. Thus, condition 1 is satisfied. Second, the execution time for processing one page of data is independent of page size, as was the case for the max operation.

Now consider how this operation would be implemented using a conventional memory. The value used as a comparand in the associative memory would be compared to each of the M data. Those data whose value is larger than this comparand value would be marked or moved to a special location.

This discussion shows that M comparisons would be required and, therefore, the execution time is proportional to M. The ratio of execution time for the associative memory implementation is the following.

$$\frac{t\,({\mathcal{P}'}_{gt'_{am}})}{t\,({\mathcal{P}'}_{gt'_{cm}})} = \frac{K_2}{M}$$

or

$$t\,({\mathcal{P}'}_{gt'_{am}}) = \frac{K_2}{M}\; t\,({\mathcal{P}'}_{gt'_{cm}})$$

where K_2 is some constant. Thus, condition 2 is satisfied. The discussions for the operations, "exact match," and "less than" are completely similar to that for gt. For this reason they will not be repeated here.

ACKNOWLEDGMENTS

The author would like to express his gratitude to Professor A. W. Naylor for valuable discussions, and to his colleague Juan Figueras for many helpful comments and suggestions.

This work was sponsored by the Supporting Developments Technical Area, U.S. Army Electronic Warfare Laboratory, under the direction of Mr. Samuel Stiber with Mr. E. Barshay serving as project engineer. Correspondence relative to this project should be addressed to: Commanding General, U.S. Army Electronics Command, ATTN: AMSEL-WL-S (Mr. E. Barshay), Fort Monmouth, New Jersey 07703.